FIT3077 - Sprint 4

CL_Monday06pm_Team003

Alex Ung, Romal Patel, Hirun Hettigoda

Table of Contents

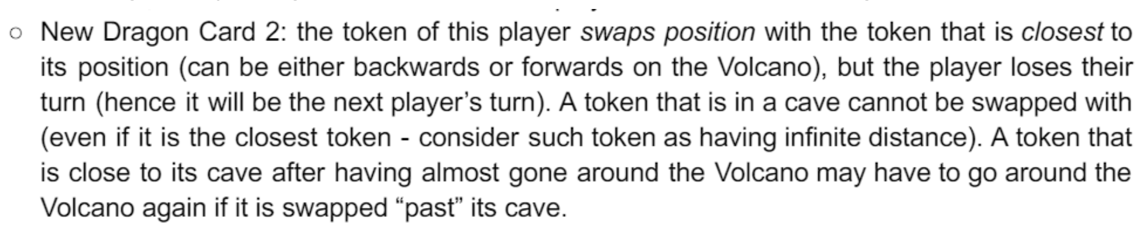
Updated UML Diagram

Executable Instructions

Reflection on Sprint 3 Design

Required Extension:

Implementing a NinjaDragon which performs the effect of the following:



- New Dragon Card 2: the token of this player *swaps position* with the token that is *closest* to its position (can be either backwards or forwards on the Volcano), but the player loses their turn (hence it will be the next player's turn). A token that is in a cave cannot be swapped with (even if it is the closest token - consider such token as having infinite distance). A token that is close to its cave after having almost gone around the Volcano may have to go around the Volcano again if it is swapped "past" its cave.

Ease of implementation:
Overall, the extension was not very difficult to implement. To implement this new dragon card, a new jPanel needed to be added to the dragonCardPool JPanel. The most difficult part was figuring out how to incorporate the ability for a dragonCard to perform a special action that is not constant like the actions of the existing dragonCard creatures. In this particular scenario, a new interface was created for these specialCreatures that perform special actions. These special creatures implement the required method in the interface. This allows for special creatures to be addressed separately from the other creatures.

However, although the extension was easy to implement, there is a presence of not so good oop practices. In this particular case, downcasting had to be used to get access to the special action of the NinjaDragon dragonCard. As a result, this is not in line with the LSP principle where the objects of the superclass should be replaceable of its subclasses without affecting the correctness of the program. In addition, the open/closed principle was violated by having to add the need to modify the existing code to account for "specialCreatures" which could be avoided.

What would the team do differently:
Incorporate polymorphism. Alter the creature class to have an abstract method "perform action" that each creature can implement. This will make things easier when calling the action of the dragon card because the action will be taken care of by the creature itself. In order to not have to repeat overlapping actions, the movement manager already has a method move which can be called pretty easily. This will also remove the need for identifying whether a dragon card is a good, bad or special which will make applying actions easier when flipping a dragon card. This would also make the code follow the open/closed principle as the existing code wouldn't have needed to be changed as adding another dragon card with its own action would be all that is required.